Syntax of FOL: Basic elements

- **Constants**: KingJohn, 2, UCB, ...
- **Predicates**: Brother, >, ...
- **Functions**: Sqrt, LeftLegOf, ...
- **Variables**: x, y, a, b, ...
- **Connectives**: ∧, ∨, ¬, ⇒, ⇔
- **Equality**: =
- **Quantifiers**: ∀, ∃

Complex sentences are made from atomic sentences using connectives

- ¬S, S₁ ∧ S₂, S₁ ∨ S₂, S₁ ⇒ S₂, S₁ ⇔ S₂

E.g., Sibling(KingJohn, Richard) ⇒ Sibling(Richard, KingJohn)
- >(1, 2) ∨ ≤(1, 2)
- >(1, 2) ∧ ¬>(1, 2)

E.g., Brother(KingJohn, RichardTheLionheart)
- > (Length(LeftLegOf(Richard)), Length(LeftLegOf(KingJohn)))
Truth in first-order logic

Sentences are true with respect to a model and an interpretation.

Model contains objects and relations among them. Interpretation specifies referents for:
- constant symbols → objects
- predicate symbols → relations
- function symbols → functional relations

An atomic sentence \(\text{predicate}(\text{term}_1, \ldots, \text{term}_n)\) is true iff the objects referred to by \(\text{term}_1, \ldots, \text{term}_n\) are in the relation referred to by \(\text{predicate}\).

Models for FOL: Example

- objects
  - [Diagram of objects]

- relations: sets of tuples of objects
  - \(\{ \langle \text{person}, \text{king} \rangle, \langle \text{king}, \text{king} \rangle, \ldots \} \)

- functional relations: all tuples of objects + "value" object
  - \(\{ \langle \text{person}, \_ \rangle, \langle \text{king}, \_ \rangle, \ldots \} \)
Universal quantification

\[ \forall \langle \text{variables} \rangle \langle \text{sentence} \rangle \]

Everyone at Berkeley is smart:
\[ \forall x \, \text{At}(x, \text{Berkeley}) \Rightarrow \text{Smart}(x) \]

\[ \forall x \, P \] is equivalent to the conjunction of instantiations of \( P \)

\[ \text{At}(\text{KingJohn}, \text{Berkeley}) \Rightarrow \text{Smart}(\text{KingJohn}) \]
\[ \land \text{At}(\text{Richard}, \text{Berkeley}) \Rightarrow \text{Smart}(\text{Richard}) \]
\[ \land \text{At}(\text{Berkeley}, \text{Berkeley}) \Rightarrow \text{Smart}(\text{Berkeley}) \]
\[ \land \ldots \]

Typically, \( \Rightarrow \) is the main connective with \( \forall \).
Common mistake: using \( \land \) as the main connective with \( \forall \):
\[ \forall x \, \text{At}(x, \text{Berkeley}) \land \text{Smart}(x) \]
means "Everyone is at Berkeley and everyone is smart".

Existential quantification

\[ \exists \langle \text{variables} \rangle \langle \text{sentence} \rangle \]

Someone at Stanford is smart:
\[ \exists x \, \text{At}(x, \text{Stanford}) \land \text{Smart}(x) \]

\[ \exists x \, P \] is equivalent to the disjunction of instantiations of \( P \)

\[ \text{At}(\text{KingJohn}, \text{Stanford}) \land \text{Smart}(\text{KingJohn}) \]
\[ \lor \text{At}(\text{Richard}, \text{Stanford}) \land \text{Smart}(\text{Richard}) \]
\[ \lor \text{At}(\text{Stanford}, \text{Stanford}) \land \text{Smart}(\text{Stanford}) \]
\[ \lor \ldots \]

Typically, \( \land \) is the main connective with \( \exists \).
Common mistake: using \( \Rightarrow \) as the main connective with \( \exists \):
\[ \exists x \, \text{At}(x, \text{Stanford}) \Rightarrow \text{Smart}(x) \]
is true if there is anyone who is not at Stanford!
Properties of quantifiers

∀ x ∀ y is the same as ∀ y ∀ x (why??)
∃ x ∃ y is the same as ∃ y ∃ x (why??)
∃ x ∀ y is not the same as ∀ y ∃ x
∃ x ∀ y Loves(x, y)
“There is a person who loves everyone in the world”
∀ y ∃ x Loves(x, y)
“Everyone in the world is loved by at least one person”

Quantifier duality: each can be expressed using the other
∀ x Likes(x, IceCream) →∃ x ¬Likes(x, IceCream)
∃ x Likes(x, Broccoli) →∀ x ¬Likes(x, Broccoli)
◊ Fun with sentences

Equality

term₁ = term₂ is true under a given interpretation if and only if term₁ and term₂ refer to the same object
E.g., 1 = 2 and ∀ x ×(Sqrt(x), Sqrt(x)) = x are satisfiable
2 = 2 is valid

E.g., definition of (full) Sibling in terms of Parent:
∀ x, y Sibling(x, y) ⇔ [(¬(x = y) ∧ ∃ m, f ¬(m = f) ∧ Parent(m, x) ∧ Parent(f, x) ∧ Parent(m, y) ∧ Parent(f, y)]
Interacting with FOL KBs

Suppose a wumpus-world agent is using an FOL KB and perceives a smell and a breeze (but no glitter) at $t = 5$:

\[ \text{TELL}(KB, \text{Percept}([\text{Smell}, \text{Breeze}, \text{None}], 5)) \]
\[ \text{ASK}(KB, \exists a \: \text{Action}(a, 5)) \]

I.e., does the KB entail any particular actions at $t = 5$?

Answer: Yes, \{a/Shoot\} ← substitution (binding list)

Given a sentence $S$ and a substitution $\sigma$,

$S\sigma$ denotes the result of plugging $\sigma$ into $S$; e.g.,

$S = \text{Smarter}(x, y)$
$\sigma = \{x/\text{Hillary}, y/\text{Bill}\}$
$S\sigma = \text{Smarter}(\text{Hillary}, \text{Bill})$

\[ \text{ASK}(KB, S) \text{ returns some/all } \sigma \text{ such that } KB \models S\sigma \]

Knowledge base for the wumpus world

"Perception"

$\forall b, g, t \: \text{Percept}([\text{Smell}, b, g], t) \Rightarrow \text{Smelt}(t)$
$\forall s, b, t \: \text{Percept}([s, b, \text{Glitter}], t) \Rightarrow \text{AtGold}(t)$

Reflex: $\forall t \: \text{AtGold}(t) \Rightarrow \text{Action}(\text{Grab}, t)$

Reflex with internal state: do we have the gold already?

$\forall t \: \text{AtGold}(t) \land \neg \text{Holding}(\text{Gold}, t) \Rightarrow \text{Action}(\text{Grab}, t)$

$\text{Holding}(\text{Gold}, t)$ cannot be observed

⇒ keeping track of change is essential
Deducing hidden properties

Properties of locations:
\[ \forall l, t \; \text{At}(\text{Agent}, l, t) \land \text{Smelt}(t) \Rightarrow \text{Smelly}(l) \]
\[ \forall l, t \; \text{At}(\text{Agent}, l, t) \land \text{Breeze}(t) \Rightarrow \text{Breezy}(l) \]

Squares are breezy near a pit:

Diagnostic rule—infer cause from effect
\[ \forall y \; \text{Breezy}(y) \Rightarrow \exists x \; \text{Pit}(x) \land \text{Adjacent}(x, y) \]

Causal rule—infer effect from cause
\[ \forall x, y \; \text{Pit}(x) \land \text{Adjacent}(x, y) \Rightarrow \text{Breezy}(y) \]

Neither of these is complete—e.g., the causal rule doesn’t say whether squares far away from pits can be breezy

Definition for the Breezy predicate:
\[ \forall y \; \text{Breezy}(y) \iff [\exists x \; \text{Pit}(x) \land \text{Adjacent}(x, y)] \]

Keeping track of change

Facts hold in situations, rather than eternally
E.g., \text{Holding}(\text{Gold, Now}) rather than just \text{Holding}(\text{Gold})

Situation calculus is one way to represent change in FOL:
Admits a situation argument to each non-eternal predicate
E.g., \text{Now} in \text{Holding}(\text{Gold, Now}) denotes a situation

Situations are connected by the Result function
\text{Result}(a, s) is the situation that results from doing \( a \) is \( s \)
Describing actions I

“Effect” axiom—describe changes due to action
∀s AtGold(s) ⇒ Holding(Gold, Result(Grab, s))

“Frame” axiom—describe non-changes due to action
∀s HaveArrow(s) ⇒ HaveArrow(Result(Grab, s))

Frame problem: find an elegant way to handle non-change
(a) representation—avoid frame axioms
(b) inference—avoid repeated “copy-overs” to keep track of state

Qualification problem: true descriptions of real actions require endless caveats—what if gold is slippery or nailed down or . . .

Ramification problem: real actions have many secondary consequences—what about the dust on the gold, wear and tear on gloves, . . .

Describing actions II

Successor-state axioms solve the representational frame problem

Each axiom is “about” a predicate (not an action per se):

\[ P \text{ true afterwards} ⇔ \begin{cases} \text{an action made } P \text{ true} \\ \lor \text{P true already and no action made } P \text{ false} \end{cases} \]

For holding the gold:
∀a, s Holding(Gold, Result(a, s)) ⇔
\[ [(a = \text{Grab} \land \text{AtGold}(s)) \lor (\text{Holding}(Gold, s) \land a \neq \text{Release})] \]
Making plans

Initial condition in KB:
- $At(Agent, [1, 1], S_0)$
- $At(Gold, [1, 2], S_0)$

Query: $\text{Ask}(KB, \exists s \ \text{Holding}(Gold, s))$
- i.e., in what situation will I be holding the gold?

Answer: $\{s/\text{Result}(\text{Grab}, \text{Result}(\text{Forward}, S_0))\}$
- i.e., go forward and then grab the gold

This assumes that the agent is interested in plans starting at $S_0$ and that $S_0$ is the only situation described in the KB

Making plans: A better way

Represent plans as action sequences $[a_1, a_2, \ldots, a_n]$

$\text{PlanResult}(p, s)$ is the result of executing $p$ in $s$

Then the query $\text{Ask}(KB, \exists p \ \text{Holding}(Gold, \text{PlanResult}(p, S_0)))$
has the solution $\{p/\text{[Forward, Grab]}\}$

Definition of $\text{PlanResult}$ in terms of $\text{Result}$:
- $\forall s \ \text{PlanResult}([], s) = s$
- $\forall a, p, s \ \text{PlanResult}([a|p], s) = \text{PlanResult}(p, \text{Result}(a, s))$

Planning systems are special-purpose reasoners designed to do this type of inference more efficiently than a general-purpose reasoner.
Summary

First-order logic:
- objects and relations are semantic primitives
- syntax: constants, functions, predicates, equality, quantifiers

Increased expressive power: sufficient to define wumpus world

Situation calculus:
- conventions for describing actions and change in FOL
- can formulate planning as inference on a situation calculus KB

Inference in first-order logic

◊ Chapter 9, Sections 1–4
◊ Proofs
◊ Unification
◊ Generalized Modus Ponens
◊ Forward and backward chaining
Proofs

Sound inference: find $\alpha$ such that $KB \models \alpha$.
Proof process is a search, operators are inference rules.

E.g., Modus Ponens (MP)

$$
\frac{\alpha, \alpha \Rightarrow \beta}{\beta} \quad \frac{At(Joe, UCB)}{OK(Joe)} \quad \frac{At(Joe, UCB)}{OK(Joe)}
$$

E.g., And-Introduction (AI)

$$
\frac{\alpha \beta}{\alpha \land \beta} \quad \frac{OK(Joe) \quad CSMajor(Joe)}{OK(Joe) \land CSMajor(Joe)}
$$

E.g., Universal Elimination (UE)

$$
\frac{\forall x \alpha}{\alpha\{x/\tau\}} \quad \frac{\forall x \ At(x, UCB) \Rightarrow OK(x)}{At(Pat, UCB) \Rightarrow OK(Pat)}
$$

$\tau$ must be a ground term (i.e., no variables)

Example proof

| Bob is a buffalo | 1. Buffalo(Bob) |
| Pat is a pig | 2. Pig(Pat) |
| Buffaloes outrun pigs | 3. $\forall x, y \ Buffalo(x) \land Pig(y) \Rightarrow Faster(x, y)$ |
| Bob outruns Pat |  |
Search with primitive inference rules

Operators are inference rules
States are sets of sentences
Goal test checks state to see if it contains query sentence

AI, UE, MP is a common inference pattern

Problem: branching factor huge, esp. for UE

Idea: find a substitution that makes the rule premise match some known facts
⇒ a single, more powerful inference rule

Unification

A substitution $\sigma$ unifies atomic sentences $p$ and $q$ if $p\sigma = q\sigma$

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(\text{John}, \text{Jane})$</td>
<td></td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(y, \text{OJ})$</td>
<td></td>
</tr>
<tr>
<td>$\text{Knows}(\text{John}, x)$</td>
<td>$\text{Knows}(y, \text{Mother}(y))$</td>
<td></td>
</tr>
</tbody>
</table>

Idea: Unify rule premises with known facts, apply unifier to conclusion

E.g., if we know $q$ and then we conclude

$\text{Knows}(\text{John}, x) \Rightarrow \text{Likes}(\text{John}, x)$
$\text{Likess}(\text{John}, \text{Jane})$
$\text{Likes}(\text{John}, \text{OJ})$
$\text{Likes}(\text{John}, \text{Mother}(\text{John}))$
Generalized Modus Ponens (GMP)

\[ \frac{p_1', p_2', \ldots, p_n', (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q)}{q\sigma} \]  

where \( p_i'\sigma = p_i\sigma \) for all \( i \)

E.g. \( p_1' = \text{Faster}(\text{Bob}, \text{Pat}) \)
\( p_2' = \text{Faster}(\text{Pat}, \text{Steve}) \)

\( p_1 \land p_2 \Rightarrow q = \text{Faster}(x, y) \land \text{Faster}(y, z) \Rightarrow \text{Faster}(x, z) \)
\( \sigma = \{x/\text{Bob}, y/\text{Pat}, z/\text{Steve}\} \)
\( q\sigma = \text{Faster}(\text{Bob}, \text{Steve}) \)

GMP used with KB of definite clauses (exactly one positive literal):
either a single atomic sentence or
(conjunction of atomic sentences) \( \Rightarrow \) (atomic sentence)
All variables assumed universally quantified

Soundness of GMP

Need to show that

\[ p_1', \ldots, p_n', (p_1 \land \ldots \land p_n \Rightarrow q) \models q\sigma \]

provided that \( p_i'\sigma = p_i\sigma \) for all \( i \)

Lemma: For any definite clause \( p \), we have \( p \models p\sigma \) by UE

1. \( (p_1 \land \ldots \land p_n \Rightarrow q) \models (p_1 \land \ldots \land p_n \Rightarrow q)\sigma = (p_1\sigma \land \ldots \land p_n\sigma \Rightarrow q\sigma) \)
2. \( p_1', \ldots, p_n' \models p_1' \land \ldots \land p_n' \models p_1'\sigma \land \ldots \land p_n'\sigma \)
3. From 1 and 2, \( q\sigma \) follows by simple MP
Forward chaining

When a new fact \( p \) is added to the KB
for each rule such that \( p \) unifies with a premise
if the other premises are known
then add the conclusion to the KB and continue chaining

Forward chaining is data-driven
  e.g., inferring properties and categories from percepts

---

Forward chaining example

Add facts 1, 2, 3, 4, 5, 7 in turn.
Number in [] = unification literal; \( \sqrt{\} \) indicates rule firing

1. Buffalo(x) \( \land \) Pig(y) \( \Rightarrow \) Faster(x, y)
2. Pig(y) \( \land \) Slug(z) \( \Rightarrow \) Faster(y, z)
3. Faster(x, y) \( \land \) Faster(y, z) \( \Rightarrow \) Faster(x, z)
4. Buffalo(Bob) [1a, \( \times \)]
5. Pig(Pat) [1b, \( \sqrt{\} \)] \( \rightarrow \) 6. Faster(Bob, Pat) [3a, \( \times \)], [3b, \( \times \)]
6a. Faster(Bob, Pat) [3a, \( \times \)], [3b, \( \times \)]
6b. Faster(Bob, Pat) [3a, \( \times \)], [3b, \( \times \)]
7. Slug(Steve) [2b, \( \sqrt{\} \)]
   \( \rightarrow \) 8. Faster(Pat, Steve) [3a, \( \times \)], [3b, \( \sqrt{\} \)]
   \( \rightarrow \) 9. Faster(Bob, Steve) [3a, \( \times \)], [3b, \( \times \)]
Backward chaining

When a query $q$ is asked
if a matching fact $q'$ is known, return the unifier
for each rule whose consequent $q'$ matches $q$
attempt to prove each premise of the rule by backward chaining

(Some added complications in keeping track of the unifiers)

(More complications help to avoid infinite loops)

Two versions: find any solution, find all solutions

Backward chaining is the basis for logic programming, e.g., Prolog

Backward chaining example

1. $Pig(y) \land Slug(z) \Rightarrow Faster(y,z)$
2. $Slimy(z) \land Creeps(z) \Rightarrow Slug(z)$
3. $Pig(Pat)$
4. $Slimy(Steve)$
5. $Creeps(Steve)$

\[
\begin{array}{ccc}
  & \text{Faster(Pat,Steve)} \\
 1. & \{y/Pat, z/Steve\} \\
 3. & \{\} \\
 2. & \{z/Steve\} \\
 4. & \{\} \\
 5. & \{\} \\
\end{array}
\]
Completeness in FOL

Procedure \( i \) is complete if and only if

\[ KB \vdash_{i} \alpha \quad \text{whenever} \quad KB \models \alpha \]

Forward and backward chaining are complete for Horn KBs but incomplete for general first-order logic.

E.g., from

\[
\begin{align*}
\text{PhD}(x) & \Rightarrow \text{HighlyQualified}(x) \\
\neg \text{PhD}(x) & \Rightarrow \text{EarlyEarnings}(x) \\
\text{HighlyQualified}(x) & \Rightarrow \text{Rich}(x) \\
\text{EarlyEarnings}(x) & \Rightarrow \text{Rich}(x)
\end{align*}
\]

should be able to infer \( \text{Rich}(\text{Me}) \), but FC/BC won’t do it.

Does a complete algorithm exist?
A brief history of reasoning

450 B.C. Stoics propositional logic, inference (maybe)
322 B.C. Aristotle “syllogisms” (inference rules), quantifiers
1565 Cardano probability theory (propositional logic + uncertainty)
1847 Boole propositional logic (again)
1879 Frege first-order logic
1922 Wittgenstein proof by truth tables
1930 Gödel \( \exists \) complete algorithm for FOL
1930 Herbrand complete algorithm for FOL (reduce to propositional)
1931 Gödel \( \neg \exists \) complete algorithm for arithmetic
1960 Davis/Putnam “practical” algorithm for propositional logic
1965 Robinson “practical” algorithm for FOL—resolution

Resolution

Entailment in first-order logic is only semidecidable:
- can find a proof of \( \alpha \) if \( KB \models \alpha \)
- cannot always prove that \( KB \not\models \alpha \)

Cf. Halting Problem: proof procedure may be about to terminate with success or failure, or may go on for ever

Resolution is a refutation procedure:
- to prove \( KB \models \alpha \), show that \( KB \land \neg\alpha \) is unsatisfiable

Resolution uses \( KB, \neg\alpha \) in CNF (conjunction of clauses)

Resolution inference rule combines two clauses to make a new one:

\[
\begin{tikzpicture}
\node (c1) at (0,0) {C_1};
\node (c2) at (1,0) {C_2};
\node (c) at (0.5,0) {C};
\path (c1) edge (c);
\path (c2) edge (c);
\end{tikzpicture}
\]

Inference continues until an empty clause is derived (contradiction)
Resolution inference rule

Basic propositional version:

\[ \frac{\alpha \lor \beta, \neg \beta \lor \gamma}{\alpha \lor \gamma} \quad \text{or equivalently} \quad \frac{\neg \alpha \Rightarrow \beta, \beta \Rightarrow \gamma}{\neg \alpha \Rightarrow \gamma} \]

Full first-order version:

\[ p_1 \lor \ldots \lor p_j \lor \ldots \lor p_m, \]
\[ q_1 \lor \ldots \lor q_k \lor \ldots \lor q_n \]
\[ \frac{(p_1 \lor \ldots \lor p_{j-1} \lor p_{j+1} \ldots p_m \lor q_1 \lor \ldots \lor q_{k-1} \lor q_{k+1} \ldots q_n) \sigma}{\text{where } p_j \sigma = \neg q_k \sigma} \]

For example,

\[ \neg \text{Rich}(x) \lor \text{Unhappy}(x) \]
\[ \text{Rich}(\text{Me}) \]
\[ \frac{\text{Unhappy}(\text{Me})}{\text{with } \sigma = \{x/\text{Me}\}} \]

Conjunctive Normal Form

Literal = (possibly negated) atomic sentence, e.g., \( \neg \text{Rich} (\text{Me}) \)

Clause = disjunction of literals, e.g., \( \neg \text{Rich}(\text{Me}) \lor \text{Unhappy}(\text{Me}) \)

The KB is a conjunction of clauses

Any FOL KB can be converted to CNF as follows:

1. Replace \( P \Rightarrow Q \) by \( \neg P \lor Q \)
2. Move \( \neg \) inwards, e.g., \( \neg \forall x P \) becomes \( \exists x \neg P \)
3. Standardize variables apart, e.g., \( \forall x P \lor \exists x Q \) becomes \( \forall x P \lor \exists y Q \)
4. Move quantifiers left in order, e.g., \( \forall x P \lor \exists x Q \) becomes \( \forall x \exists y P \lor Q \)
5. Eliminate \( \exists \) by Skolemization (next slide)
6. Drop universal quantifiers
7. Distribute \( \land \) over \( \lor \), e.g., \( (P \land Q) \lor R \) becomes \( (P \lor Q) \land (P \lor R) \)
**Skolemization**

\[ \exists x \, \text{Rich}(x) \text{ becomes } \text{Rich}(\text{G1}) \text{ where G1 is a new "Skolem constant"} \]

\[ \exists k \, \frac{d}{dy}(k^y) = k^y \text{ becomes } \frac{d}{dy}(e^y) = e^y \]

More tricky when \( \exists \) is inside \( \forall \)

E.g., “Everyone has a heart”

\[ \forall x \, \text{Person}(x) \Rightarrow \exists y \, \text{Heart}(y) \land \text{Has}(x,y) \]

Incorrect:

\[ \forall x \, \text{Person}(x) \Rightarrow \text{Heart}(H1) \land \text{Has}(x,H1) \]

Correct:

\[ \forall x \, \text{Person}(x) \Rightarrow \text{Heart}(H(x)) \land \text{Has}(x,H(x)) \]

where \( H \) is a new symbol (“Skolem function”)

Skolem function arguments: all enclosing universally quantified variables

---

**Resolution proof**

To prove \( \alpha \):
- negate it
- convert to CNF
- add to CNF KB
- infer contradiction

E.g., to prove \( \text{Rich}(\text{me}) \), add \( \neg \text{Rich}(\text{me}) \) to the CNF KB

\[ \neg \text{PhD}(x) \lor \text{HighlyQualified}(x) \]
\[ \text{PhD}(x) \lor \text{EarlyEarnings}(x) \]
\[ \neg \text{HighlyQualified}(x) \lor \text{Rich}(x) \]
\[ \neg \text{EarlyEarnings}(x) \lor \text{Rich}(x) \]
Resolution proof

\[ \neg \text{PhD}(x) \lor \text{HQ}(x) \quad \neg \text{HQ}(x) \lor \text{Rich}(x) \]

\[ \{ \} \]

\[ \neg \text{PhD}(x) \lor \text{Rich}(x) \quad \text{PhD}(x) \lor \text{ES}(x) \]

\[ \{ \} \]

\[ \text{Rich}(x) \lor \text{ES}(x) \quad \neg \text{ES}(x) \lor \text{Rich}(x) \]

\[ \{ \} \]

\[ \text{Rich}(x) \quad \neg \text{Rich}(Me) \]

\[ \{x/Me\} \]

Logic programming

Sound bite: computation as inference on logical KBs

<table>
<thead>
<tr>
<th>Logic programming</th>
<th>Ordinary programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Identify problem</td>
<td>Identify problem</td>
</tr>
<tr>
<td>2. Assemble information</td>
<td>Assemble information</td>
</tr>
<tr>
<td>3. Tea break</td>
<td>Figure out solution</td>
</tr>
<tr>
<td>4. Encode information in KB</td>
<td>Program solution</td>
</tr>
<tr>
<td>5. Encode problem instance as facts</td>
<td>Encode problem instance as data</td>
</tr>
<tr>
<td>6. Ask queries</td>
<td>Apply program to data</td>
</tr>
<tr>
<td>7. Find false facts</td>
<td>Debug procedural errors</td>
</tr>
</tbody>
</table>

Should be easier to debug \( \text{Capital}(\text{NewYork}, \text{US}) \) than \( x := x + 2 \)!
Prolog systems

Basis: backward chaining with Horn clauses + bells & whistles
Widely used in Europe, Japan (basis of 5th Generation project)
Compilation techniques ⇒ 10 million LIPS
Program = set of clauses = head :- literal₁, ... literalₙ.
Efficient unification by open coding
Efficient retrieval of matching clauses by direct linking
Depth-first, left-to-right backward chaining
Built-in predicates for arithmetic etc., e.g., X is Y*Z+3
Closed-world assumption (“negation as failure”)
  e.g., not PhD(X) succeeds if PhD(X) fails

Prolog examples

Depth-first search from a start state X:
dfs(X) :- goal(X).
dfs(X) :- successor(X,S),dfs(S).
No need to loop over S: successor succeeds for each
Appending two lists to produce a third:
append([],Y,Y).
append([X|L],Y,[X|Z]) :- append(L,Y,Z).

query: append(A,B,[1,2]) ?
answers: A=[]   B=[1,2]
        A=[1,2] B=[]